Interactive Cleaning for Progressive Visualization through Composite Questions

Yuyu Luo† Chengliang Chai† Xuedi Qin† Nan Tang‡ Guoliang Li†
†Tsinghua University, China ‡QCRI, Hamad Bin Khalifa University, Qatar
{luoyy18@mails., chaicl15@mails., qxd17@mails., liguoliang@}tsinghua.edu.cn, ntang@hbku.edu.qa

***Abstract*—In this paper, we study the problem of *interactive cleaning for progressive visualization* (ICPV): Given a bad visualization V, it is to obtain a "cleaned" visualization V' whose distance is *far* from V, under a given (small) budget *w.r.t.* human cost. In ICPV, a system interacts with a user iteratively. During each iteration, it asks the user a data cleaning question such as *"how to clean detected errors x?"*, and takes value updates from the user to clean V. Conventional wisdom typically picks a *single question* (*e.g.,* *"Are SIGMOD conference and SIGMOD the same?"*) with the maximum expected benefit in each iteration. We propose to use a *composite question – i.e.,* a group of single questions to be treated as one question – in each iteration (for example, *Are SIGMOD conference in t_1 and SIGMOD in t_2 the same value, and are t_1 and t_2 duplicates?*). A composite question is presented to the user as a small connected graph through a novel GUI that the user can directly operate on. We propose algorithms to select the *best* composite question in each iteration. Experiments on real-world datasets verify that composite questions are more effective than asking single questions in isolation *w.r.t.* the human cost.**

I. Introduction

Data visualization plays a key role in impacting strategic and operational decisions of today's data-driven businesses. However, data visualizations are not always *exact and good*, and the *uncertainty of data visualization* [12], [32] may misguide users by showing false discoveries [6]. One common reason for generating such bad (uncertain) visualizations is because real-life data is dirty [2].

Example 1: Table I shows publications from multiple sources. Dirty Citations cells are marked by red. Duplicated records are marked by the same color on attribute Id. Besides, it also contains unstandardized values such as "Very Large Data Bases" and "VLDB". The ground truth of Table I is given in Table II. For example, t_{123} in Table II is the consolidated record for tuples t_1, t_2, and t_3 in Table I.

[An Incorrect Bar Chart.] Figure 1(a) is a bar chart about the #-total of Citations grouped by Venue. Due to various types of errors, the visualization is incorrect. For example, duplicated bars: "SIGMOD Conf.", "ACM SIGMOD", "SIGMOD", and "SIGMOD'13" should be merged, t_2 has an *outlier* at attribute Citations that affects the bar "SIGMOD Conf.", and t_7 has a missing Citations value that affects the bar "VLDB". □

However, a visualization is not necessarily dirty, even if the data is dirty. Consider another example.

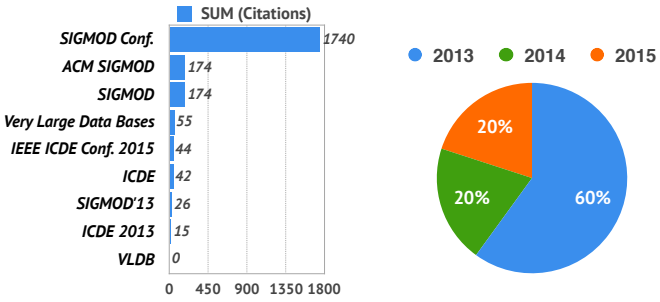
Example 2: [A Correct Pie Chart.] Figure 1(b) shows the proportion of the #-publications by Year and the result is not affected by dirty data, because the proportion of the #-publications by Year in dirty data (Table I) is the same as cleaned data (Table II). □

Id	Year	Title (abbr.)	Venue	Affiliation	Citations
t_1	2013	NADEEF	ACM SIGMOD	QCRI	174.0
t_2	2013	NADEEF	SIGMOD Conf.	QCRI, HBKU	1740
t_3	2013	NADEEF	SIGMOD	QCRI HBKU	174.0
t_4	2013	KuaFu	ICDE 2013	Microsoft	15.0
t_5	2013	TsingNUS	SIGMOD'13	Tsinghua	13.0
t_6	2013	TsingNUS	SIGMOD'13	THU	13.0
t_7	2014	SeeDB	VLDB	Stanford Univ.	N.A.
t_8	2014	SeeDB	Very Large Data Bases	Stanford	55.0
t_9	2015	Elaps	ICDE	NUS	42.0
t_{10}	2015	Elaps	IEEE ICDE Conf. 2015	CS@NUS	44.0

TABLE I
An Excerpt of Publications (Dirty)

Id	Year	Title (abbr.)	Venue	Affiliation	Citations
t_{123}	2013	NADEEF	SIGMOD	QCRI	***174.0***
t_4	2013	KuaFu	ICDE	Microsoft	15.0
t_{56}	2013	TsingNUS	SIGMOD	Tsinghua	13.0
t_{78}	2014	SeeDB	VLDB	Stantford Univ.	***55.0***
t_{910}	2015	Elaps	ICDE	NUS	43.0

TABLE II
An Excerpt of Publications (Ground Truth)



(a) An Incorrect Bar Chart (b) A Correct Pie Chart

Fig. 1. Example Visualization on Table I

Practically, it is too expensive to completely clean a dataset. Intuitively, compared with cleaning the entire dataset, only cleaning task (such as a bar chart or a pie chart) relevant data should be much cheaper.

We study a new problem, *interactive cleaning for progressive visualization* (ICPV), to progressively improve the quality of visualization by minimizing the cost of interacting with the user to clean the visualization-aware data.

Challenges. There are four main challenges. (C1) How to quantify the difference between two visualizations before/after the data is cleaned? (C2) What is the ideal user- and task-

[1]Chengliang Chai is the corresponding author.

```
VISUALIZE   TYPE in {Bar, Pie}
SELECT      X', Y' (X' in {X, GROUP(X), BIN(X)}, Y' in {Y, AGG(Y)})
FROM        D
TRANSFORM   X (using an operating in {BIN, GROUP})
SORT        X', Y' BY {DESC, ASC}
WHERE       A OP t (A in {X, Y}, OP in {=, <, <=, >=, >})
LIMIT       K
```

Fig. 2. Visualization Query Language (VQL)

friendly interface to interact with the user? (C3) How to compute the expected benefit of cleaning data errors for a task? (C4) How to select the questions that maximize the benefit of asking a set of questions?

Contributions. We develop a system, VISCLEAN, to tackle the ICPV problem, with the following contributions.

- We describe methods to quantify the difference between two visualizations (C1), characterize different types of data errors, and propose novel composite questions to interact with the users (C2). (Section II)
- We present a general framework for solving the ICPV problem. (Section III)
- We describe how to detect data errors and generate possible repairs. (Section IV)
- We propose an estimation-based benefit model to quantify the visualization quality improvement from the user interaction (C3), and an effective question selection algorithm to select the most beneficial question in each iteration (C4). (Section V)
- We present a new graph-based GUI for composite questions that users can directly operate on, for both cleaning the data and the visualization. (Section VI)
- Extensive experiments on real-world datasets show that we can effectively turn bad visualization into good ones with a small number of interactions. (Section VII)

Moreover, we discuss related work in Section VIII and close this paper by concluding remarks in Section IX.

II. PRELIMINARIES

A. Visualization Queries

We use a SQL-like visualization language [24], [25]. Other declarative languages (*e.g.,* Vega-Lite [29]) can also be used.

Figure 2 shows our used language, where the keywords are highlighted, with mandatory ones in blue and optional ones in green. X' (resp. Y') is transformed from X (resp. Y):

- X' can be just X, grouping values in X (*e.g.,* GROUP BY(Venue)), or binning values in X (*e.g.,* BIN(Citations) By Interval 200);
- Y' can be either Y or applying an aggregation function from = {SUM, AVG, COUNT} on Y.

The queries $Q_1(D)$ and $Q_2(D)$ in Fig. 3 will create a bar chart (Fig. 1(a)) and a pie chart (Fig. 1(b)), respectively.

B. Quantifying Distances between Visualizations

Let $\mathbf{dist}(Q(D), Q(D_c))$ be the *distance* between the two visualizations that use the same query Q over different versions of the same dataset, D and D_c.

```
VISUALIZE Bar
SELECT    Venue, SUM(Citations)
FROM      Table I
GROUP BY  Venue
```

$Q_1(D)$

```
VISUALIZE Pie
SELECT    Year, COUNT(Year)
FROM      Table I
GROUP BY  Year
```

$Q_2(D)$

Fig. 3. Sample Visualization Queries

Although one may use any distance function, such as Euclidean, Kullback-Leibler, Jensen-Shannon, and Earth Mover distance (EMD). In this work, we take EMD, which is known to well measure the distance between two visualizations [33].

Given two visualizations $Q(D)$ and $Q(D_c)$, let $\bar{d} = (d_1, d_2, \cdots, d_m)$ and $\bar{d'} = (d'_1, d'_2, \cdots, d'_n)$ denote the data of $Q(D)$ and $Q(D_c)$ respectively, where $d_i = (d_i(x), d_i(y))$ and $d'_i = (d'_i(x), d'_i(y))$. Let $\delta_{ij} = |d_i(y) - d'_j(y)|$ denote the distance between $d_i(y)$ and $d'_j(y)$. Note that, we normalize each $d_i(y)$ ($d'_i(y)$) into a probability distribution such that the $\sum d_i(y)$ ($\sum d'_i(y)$) is equal to 1.'' EMD aims to find a flow $F = [f_{ij}]$ that minimizes:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} \delta_{ij} \tag{1}$$

subject to the constraints:

$$f_{ij} \geq 0, \sum_{j=1}^{n} f_{ij} \leq d_i(y), \sum_{i=1}^{m} f_{ij} \leq d'_j(y) \tag{2}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} \leq \min\Big(\sum_{i=1}^{m} d_i(y), \sum_{i=1}^{n} d'_i(y)\Big) \tag{3}$$

Then, EMD is computed as:

$$EMD(Q(D), Q(D_c)) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} \delta_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij}} \tag{4}$$

C. Data Errors Meet Visualizations

Data Errors. We consider the following types of errors: (i) *tuple-level duplicates* such as t_7 and t_8 in Table I, (ii) *attribute-level duplicates* (or synonyms), such as "VLDB", and "Very Large Data Bases", (iii) *missing values* such as t_7[Citations], and (iv) *outliers* such as t_2[Citations].

Note that other types of errors can also be incorporated, if the tools for detecting and repairing them are available.

Visualizations with Data Errors. Consider the followings.

(i) Tuple-level duplicates will affect a visualization because (1) the tuples should be grouped together in X' if they refer to the same tuples, and (2) the aggregated value in Y' should be generated based on the same entity, and the *COUNT*/*SUM* functions on Y will be significantly affected. Consider $Q_1(D)$ in Fig. 3. The Citations of t_6 is counted twice, which leads to an incorrect visualization in Fig. 1(a).

(ii) Attribute-level duplicates will affect X' and Y'. (1) The values in X should be grouped based on the same attribute-level entity. (2) If X' is generated from X by some selection operations, some tuples may be missing in X' because they do not exactly match the selection conditions due to the synonym.

#	Query Type	Tuple	Attr.	Missing	Outliers
1	X'=X(Numeric), $Y'=Y$	Yes	No	Yes	Yes
2	X'=X(Category), $Y'=Y$	Yes	Yes	Yes	Yes
3	X'=*BIN*(X), Y'=*AGG*(Y)	Yes	No	Yes	Yes
4	X'=*GROUP*(X), Y'=*AGG*(Y)	Yes	Yes	Yes	Yes

TABLE III
RELATIONSHIP BETWEEN ERRORS AND VISUALIZATIONS

(3) The aggregated value in Y' will be incorrect due to errors in X'. Fig. 1(a) is an incorrect visualization that is partially caused by attribute-level duplicates.

(iii) Missing values will affect X' and Y'. (1) X' will be incorrectly grouped/binned/selected due to missing values. (2) Y' will be incorrectly aggregated. For example, t_7[Citations] is a missing value. It affects the aggregation result of the "VLDB" group in Fig. 1(a).

(iv) Outliers will affect X' and Y'. (1) If X' is selected based on selection operations or binning on X, there may be some errors due to outliers. (2) Y' will be incorrectly aggregated due to outliers such as t_2[Citations]. It greatly affects the aggregation result of "SIGMOD Conf." group in Fig. 1(a).

Table III summarizes the relationships between different types of errors and visualizations. We assume that Y-axis is always a numerical column. For example, the $Q_1(D)$ in Fig. 3 belongs to the fourth visualization query type in Table III. The X of $Q_1(D)$ is Venue, and the Y of $Q_1(D)$ is Citations. The four types of data errors will affect the quality of visualization $Q_1(D)$, as discussed above.

D. Single Questions vs. Composite Questions

Single Questions. Let's first discuss single questions.

(i) Tuple-level duplicates: "*Are t_1 and t_2 duplicates*", where tuples t_1 and t_2 are possible duplicates.

(ii) Attribute-level duplicates: *Are value v_1* (e.g., ICDE) *and v_2* (e.g., ICDE Conf.) *the same? If so, which value should be used?*

(iii) Missing values: *Which possible repair a missing value (attribute A of tuple t) should take?*

(iv) Outliers: *Is value v in tuple t an outlier? If so, which (possible) repair should it take?*

We use T-, A-, M-, and O-question to denote a tuple-level duplicates question, attribute-level duplicates question, a missing value question, and an outlier question, respectively.

Composite Questions. In practice, it is hard for a user to precisely answer a data cleaning question without enough context. Consequently, interactive data cleaning systems typically provide more information to the user, beyond only one data error [9]. Moreover, it has also observed that different errors may impact each other [14]. Hence, we propose to use one graph model to organize different types of errors.

Definition 2.1: [Errors and Repairs Graph (ERG).] All errors and possible repairs can be modeled as an undirected weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where:

1) each vertex $v_i \in \mathcal{V}$ is a tuple,

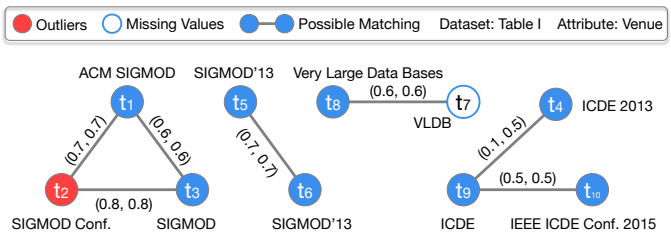
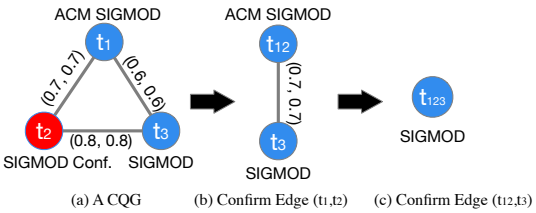


Fig. 4. A Sample ERG



(a) A CQG (b) Confirm Edge (t_1,t_2) (c) Confirm Edge (t_{12},t_3)

Fig. 5. Sample Operations on a CQG

2) there is an edge $(v_i, v_j) \in \mathcal{E}$, if v_1 and v_2 are possible tuple- or attribute-level duplicates.
3) each vertex has a label, where red means that the tuple has an outlier (*i.e.,* an O-questoin), and hollow indicates a missing value (*i.e.,* an M-question), and
4) each edge (v_i, v_j) has a weight (p^t_{ij}, p^a_{ij}), where p^t_{ij} is the tuple-level entity matching probability (*i.e.,* a T-question) and p^a_{ij} is the attribute-level matching probability (*i.e.,* an A-question). □

Note that, an A-question is only for categorical values, *i.e.,* for the X-axis of a visualization. Hence, typically, there has at most one A-question for an edge.

Example 3: [ERG.] Figure 4 is a sample ERG for Fig. 1(a). It has 10 vertices, one contains an O-question (t_2) and another has an M-question (t_7). Note that it just shows edges that are weighted (either p^t_{ij} or p^a_{ij}) between 0.5 and 0.8. It has 7 edges to denote 7 T-questions and 7 A-questions. Particularly, the edge (t_4, t_9) denotes that entity t_4 and t_9 are tending to match with probability 0.1, while the attribute values (*i.e.,* Venue) for X on t_4 and t_9 match with probability 0.5. □

Instead of using single questions on an ERG to interact with the user, we propose to use **composite questions** to provide more context for the user, which is also considered in [5].

Definition 2.2: [CQG.] A *composite question graph* is a connected induced subgraph of an ERG. □

User Permissible Operations on a CQG. *For each edge*, the user can *confirm* (resp. *split*) an edge to denote its associated two vertices are (resp. not) the same tuple- or attribute-level entity. *For each vertex*, the user can either *approve* or *reject* the outliers and missing values repair candidates.

Example 4: [CQG.] Figure 5(a), a subgraph of Fig. 4, is a CQG. The user can directly operate on Fig. 5(a) to provide label data. For example, the user *confirms* the edge (t_1, t_2) and approves the outlier repair value on t_2. It will merge vertices t_1 and t_2 (Fig. 5(b)), which denotes that t_1 and t_2 are the

same tuple- and attribute-level entity. The user further *confirms* the edge (t_{12}, t_3), similarly, it will produce Fig. 5(c). So far, it indicates the vertices (*i.e.,* tuples) t_1, t_2, and t_3 are the same entity, standardizes the variant values of "SIGMOD", and approves the outlier repairing value on t_2. These label data will enhance the quality of data cleaning models and thus improve the visualization quality. □

E. Goal

Given a bad visualization $Q(D)$ over the dirty dataset D, and a user budget B for the number of iterations that a user can be involved, the problem of *interactive cleaning for progressive visualization* (ICPV) is to interact with the user to repair D to get D_c up to B interactions, so as to maximize $\mathbf{dist}(Q(D), Q(D_c))$.

Remark. The problem of ICPV can be generalized to other types of visualizations or other task-driven data cleaning problems, if a distance function can be defined to quantify the impact on a visualization (or task) before/after cleaning a dataset. Besides the contribution of proposing composite questions versus traditional single questions, we use data visualization because it is easy for readers to understand the difference perceptually.

III. Framework Overview

The overview of VisClean for solving the ICPV problem is given in Fig. 6, with the following steps.

(1) Visualization Specification. The user needs to specify a visualization query on a specific dataset.

(2) Initialization. VisClean first needs to run off-the-shelf data cleaning tools to detect different types of errors and generate possible repairs (Section IV).

(3) ERG Construction. It then builds an ERG to organize the detected data errors and their possible repairs.

(4) CQG Selection. It first measures the benefit of asking a CQG by an estimation-based benefit model, where the benefit is associated with the distance between visualization before/after cleaning the data. The larger the distance is, the larger the benefit is (Section V-A). It then selects the "most beneficial" CQG from the ERG based on the benefit model to interact with the user in each iteration (Section V-B).

(5) User Interaction on Graph. The user interacts with our GUI to provide the answer to the CQG (Section **??**).

(6) Repair Errors and Update Visualization. After getting feedback from the user, it will repair data errors and refresh the data visualization **(7)**. Afterwards, the user feedback might be used by each used data cleaning model to find new errors/possible repairs **(2)**. The process iterates until the budget is used up.

IV. Data Errors and Possible Repairs

In this section, we will describe how to detect data errors and generate possible repairs to clean visualization.

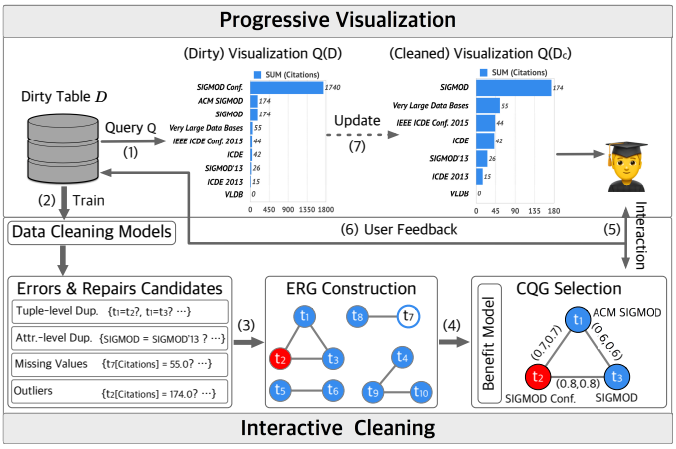


Fig. 6. Solution Overview

$\mathbb{Q}_T$: Questions for Tuple-Level Duplicates. We utilize entity matching (EM) techniques to detect and remove the duplicates. We first train an EM model (we use random forests [19]). For each tuple pair (*e.g.,* t_1 and t_3), the EM model will provide a matching probability. Then we use the active learning techniques to generate a set of tuple pairs $\mathbb{Q}_T$, *e.g.,* those uncertain pairs with probability close to 0.5, and take the pairs in $\mathbb{Q}_T$ as T-questions. A T-question for Table I is "(t_1, t_3): whether t_1 and t_3 are the same entity or not".

$\mathbb{Q}_A$: Questions for Attribute-Level Duplicates. If X is a categorical column (*e.g.,* Venue), we need to detect whether there exist attribute-level duplicates in this column, as shown in Tabe III. We have two strategies to detect attribute-level duplicates and generate a set of A-questions $\mathbb{Q}_A$.

Strategy 1. We can use T-questions to generate a set of A-questions. We group those matching tuple pairs into the same cluster. The attribute of each tuple in the same cluster should refer to the same entity. For example, t_7 and t_8 will be grouped into the same cluster, and thus t_7[Venue] and t_8[Venue] should be the same. Therefore, we can generate an A-question "Very Large Data bases"↔"VLDB". We implement this strategy based on the existing techniques *GoldenRecordCreation* [10]. For example, given a cluster C_1 : $\{t_1, t_2, t_3\}$ and attribute Venue as inputs, *GoldenRecordCreation* will produce three transformation candidates for C_1 on attribute Venue: "ACM SIGMOD"↔"SIGMOD Conf.", "ACM SIGMOD"↔"SIGMOD", and "SIGMOD Conf."↔SIGMOD". Therefore, we can standardize the variant values in each cluster.

However, if some attribute-level duplicates could not be grouped into the same cluster, it's hard for *GoldenRecordCreation* to generate all transformation candidates. For example, consider cluster C_1 : $\{t_1, t_2, t_3\}$ and C_2 : $\{t_5, t_6\}$, it can not generate some transformation candidates, *e.g.,* "SIGMOD'13"↔"SIGMOD" across clusters C_1 and C_2.

Strategy 2. We propose another strategy to tackle the above problem. Given two clusters C_i and C_j, the basic idea is that we run a *String Similarity Join* algorithm [16] on the target column X (*e.g.,*Venue) of these two clusters to find a set of similarity strings$\{(c_i, c_j) | c_i \in C_i, c_j \in C_j, \text{similarity}(c_i, c_j) > \lambda\}$.

Algorithm 1: A-QUESTIONSGENERATION

Input: Clusters: $\mathbf{C}$:$\{C_1, ..., C_n\}$, Attribute: a, Threshold: λ;
Output: A set of A-questions: $\mathbb{Q}_A$;

```
Q_A = GoldenRecordCreation(C, a);
for each C_i in C do
    for each C_j in C do
        if C_i ≠ C_j then
            Q_A = StringSimilarityJoin(C_i, C_j, a, λ);
            Q_A.append(Q_A); // add Q_A into Q_A
return Q_A;
```

For example, given two clusters $\{t_1, t_2, t_3\}$ and $\{t_5, t_6\}$, it could produce our preferred transformation candidate: "SIGMOD'13"↔"SIGMOD".

Hence, we can combine the above two strategies to generate a set of A-questions $\mathbb{Q}_A$. Algorithm 1 shows the pseudo-code.

$\mathbb{Q}_M$: Questions for Missing Values Imputation. Given a tuple with a missing value on column Y (*e.g.,* Citations), we compute k (*e.g.,* $k = 5$) nearest neighbors of the tuple. In order to measure the distance between each pair of tuples, we concatenate all attributes in each tuple to form a string and then utilize the string similarity score (*e.g.,* Jaccard) as the distance. Then, we use the average of values on column Y of these k neighbors as the suggested imputation value. Thus, for all tuples that contain missing values, we can get a set of M-questions $\mathbb{Q}_M$.

$\mathbb{Q}_O$: Questions for Outliers Repairing. We use kNN [28] to detect outliers on column Y (*e.g.,* Citations), which computes an outlier score for each value v. The score is defined as the k-th smallest absolute difference between all other values and v. Then we sort all the values based on the scores in descending order. Next, we select those values with the largest scores as the O-questions $\mathbb{Q}_O$.

Thus, we could generate a set of data repairing candidates in each iteration. Formally, let $\mathbb{Q} = \mathbb{Q}_T \cup \mathbb{Q}_A \cup \mathbb{Q}_M \cup \mathbb{Q}_O$ denote the repairing candidate set in each iteration. These questions will be organized as an ERG.

V. SELECTING COMPOSITE QUESTIONS

A. Estimation-based Benefit Model

We first discuss how to measure the benefit on an edge of a CQG. The user can *confirm/split* an edge to provide new training data to enhance the quality of data cleaning models and thus improve the quality of visualization. More specifically, for each edge (*i.e.,* each repairing candidate $q \in \mathbb{Q}$), we enumerate all possible user operations in this edge (*i.e., confirm/split* operation). For each type of user operation, it provides useful feedback to repair data errors and retrain data cleaning models (Step-**(6)** in Fig. 6). Then we can derive a new version of dataset D_c from D. At last, we can update the visualization on D_c (Step-**(7)** in Fig. 6). Thus, we can measure the visualization distance **dist** before/after cleaning the data using EMD function. The larger the distance is, the larger the estimated benefit is.

Next, we discuss how to quantify the benefit of a CQG. We assume that the user operation on each edge is independent. Therefore, we use the sum of total benefit of each edge to estimate the benefit of a CQG.

Definition 5.1: [**Estimation-based Benefit Model**] *Consider a CQG $G(V, E)$. For each edge (v_i, v_j) in E, let $\mathbf{P}^Y$ be the probability of* confirm *edge and $\mathbf{P}^N$ be the probability of* split *edge. For each type of operation, it will enhance data cleaning models and thus derive a new visualization. Hence, we use the* $\mathbf{dist}$ *to measure the visualization distance. Thus, the expected benefit of asking a CQG question could be computed as below:*

$$\mathcal{B}(G) = \sum_{i=1}^{|E|} (\mathbf{P}_i^Y \mathbf{dist}_i^Y + \mathbf{P}_i^N \mathbf{dist}_i^N) \tag{5}$$

□

Next, we discuss how to compute the estimation-based benefit for each type of repairing question.

(1) Questions for Tuple-level Duplicates. Given a tuple pair T-question in $\mathbb{Q}_T$ to be labeled (*i.e.,* an edge in the CQG), it will be answered as a matching (*i.e.,* confirm edge) or non-matching pair (*i.e.,* split edge) by the user. We use the current EM model to predict matching probability $\mathbf{P}^Y$ for tuple pairs. The probability of a matching tuple pair is $\mathbf{P}^Y$ and non-matching $1 - \mathbf{P}^Y$. (*i*) If the user confirms a T-question, the benefit is twofold. On the one hand, the EM Model can benefit from the newly labeled tuple pairs. On the other hand, these labeled tuple pairs can be used to standardize the attribute-level duplicates. For example, suppose t_7 and t_8 refer to the same entity, and thus their attribute-level entity t_7[Venue] and t_8[Venue] also refer to the same entity. Therefore, we can compute a visualization distance $\mathbf{dist}^Y$ before/after updating the EM model. (*ii*) If the user label a T-question as non-matching pair, the label answer only benefits the EM model. Similarly, we can compute a visualization distance $\mathbf{dist}^N$. Finally, the estimated benefit of a T-question is computed as:

$$\mathcal{B}_T = \mathbf{P}^Y \mathbf{dist}^Y + (1 - \mathbf{P}^Y) \mathbf{dist}^N \tag{6}$$

(2) Questions for Attribute-level Duplicates. For each A-question in $\mathbb{Q}_A$, the user may *approve* or *reject* it. We use the similarity score from Algorithm 1 to predict the probability $\mathbf{P}^Y$ that the user *approves* an A-question. (I) If the user approves an A-question, its benefit is twofold: (*i*) the benefit from standardizing attribute-level duplicates, and (*ii*) the benefit from enhancing the EM model for better removing tuple-level duplicates. We first update the entire dataset based on the answer of the A-question and then apply the answer of the A-question on the training set of the EM model. Then, we retrain the EM model and derive a new visualization. Next, we compute the visualization distance $\mathbf{dist}^Y$. (II) If the user rejects an A-question, it will not provide any benefit for improving the quality of visualization. Therefore, the overall estimated benefit for an A-question is $\mathcal{B}_A = \mathbf{P}^Y \mathbf{dist}^Y$.

(3) Questions for Missing Values Imputation. The user has two possible ways to answer an M-question in $\mathbb{Q}_M$. On the

Algorithm 2: GSS: GREEDYSUBGRAPHSELECTION

Input: An ERG $\mathcal{G}(\mathcal{V}, \mathcal{E}, w)$, k ($2 < k < |\mathcal{V}|$);
Output: Subgraph $G(V, E)$;

```
1  B_max ← 0; Collection of vertex set C ← null;
2  for each v in V do m[v] ← null ;
3  G(V, E, b) ← EstimatedBenefit(G(V, E, w));
4  Sort E by b in descending order;
5  for each edge (v, v') in E do
6      if m[v] = m[v'] = null then
7          Add {v, v'} to C; // Case 1
8          m[v] ← {v, v'}; m[v'] ← {v, v'};
9          continue ;
10     if m[v] = null then
11         v_f ← v; v_t ← v'; // Case 2
12     else
13         v_f ← v'; v_t ← v; // Case 3
14     Add v_f into m[v_t];
15     m[v_f] ← m[v_t];
16     if |m[v_t]| = k then
17         Get the G'(V', E') induced by vertices in m[v_t];
18         B ← sum of the benefit of all edges in G'(V', E');
19         if B > B_max then
20             G(V, E) ← G'(V', E');
21             B_max ← B;
22         for each vertex u in G'(V', E') do m[u] ← null;
23 return G(V, E);
```

one hand, the user can *approve* our suggested missing value imputation result. On the other hand, the user can provide their preferred missing value imputation result. For the above two cases, we use the suggested value to impute missing values and derive a new visualization. Then we compute the visualization distance before/after imputing missing values, denoted by $\mathbf{dist}^Y$. Therefore, the estimated benefit for an M-question is $\mathcal{B}_M = \mathbf{dist}^Y$.

(4) Questions for Outliers Repairing. Similar to missing value imputation, we use our suggested outlier repairing value to repair the outlier and compute the visualization distance as the estimated benefit for an O-question: $\mathcal{B}_O = \mathbf{dist}^Y$.

Example 5: [Estimated Benefit of a CQG.] Figure 5(a) is a CQG. We iterate each edge of the CQG to compute its benefit. First, we access the edge (t_1, t_2). We first compute $\mathcal{B}_T$ and then $\mathcal{B}_A$. Suppose the $\mathcal{B}_T = 0.1$ and $\mathcal{B}_A = 0.2$. Then we compute $\mathcal{B}_O$ because vertex t_2 contains an outlier repair candidate. Suppose the result of $\mathcal{B}_O = 0.2$. Thus, the benefit of edge (t_1, t_2), b_{12}, can be computed as $0.1+0.2+0.2 = 0.5$. Similarly, we can compute b_{13} and b_{23} for edge (t_1, t_3) and (t_2, t_3), respectively. Finally, we can sum b_{12}, b_{13}, and b_{23} to estimate the benefit of this CQG. □

B. CQG Selection Algorithm

After we know how to estimate the benefit of CQG, we now aim to select the "most beneficial" CQG from the ERG, *i.e.,* the "most beneficial" connected subgraph of the ERG. We describe the CQG selection problem below.

Definition 5.2: [Optimal CQG Selection.] *Given an ERG $\mathcal{G}(\mathcal{V}, \mathcal{E}, b)$ and an integer k ($2 < k < |\mathcal{V}|$), the optimal CQG selection problem aims to find a connected subgraph $G(V, E)$ of $\mathcal{G}$ with k vertices (k-subgraph) such that the total edge weight (*i.e., *benefit) is maximum.* □

For example, suppose that we want to select the optimal CQG with 4 vertices ($k = 4$) from the ERG (*e.g.,* Fig. 7(b)). The straightforward approach is enumerating all connected subgraphs with 4 vertices, compute total weight (*i.e.,* total benefit) of each subgraph, and then select the subgraph with maximum total weight as the optimal CQG (*i.e.,* Fig. 7(c)).

Unfortunately, the *Optimal CQG Selection* problem is NP-hard when $k = \mathcal{O}(\mathcal{V})$ because it can be formulated as the *Heaviest k-subgraph* problem [21], which is an NP-hard problem by a direct reduction from the *Clique* problem. When k is a constant, the time complexity of the *Optimal CQG Selection* problem is $\mathcal{O}(\binom{|V|}{k} \cdot k^2)$ because we need to enumerate all k-subgraphs and compute the benefit for each k-subgraph.

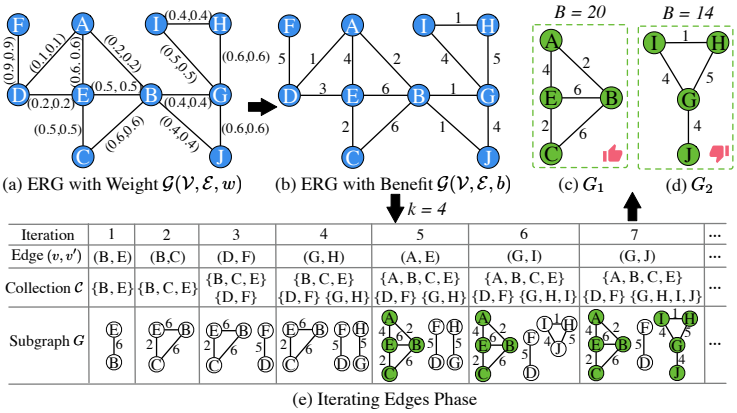
Letsios et.al. [21] developed a Branch and Bound (B&B) algorithm for *Heaviest k-subgraph* problem. However, it is much inefficient when $k > 10$, whose time complexity is $\mathcal{O}(\binom{|\mathcal{E}|}{k-1} \cdot klog(c))$, where c is the core value of the graph.

As it is prohibitively expensive to find the optimal CQG from the ERG, we devise a greedy algorithm GSS to select the CQG efficiently. The basic idea is that we select subgraphs with large benefits as candidates greedily and incrementally. Then we choose the one of size k with the largest benefit as $G(V, E)$. We first introduce some important variables in our algorithm. $\mathcal{C}$ is a collection of vertices in $\mathcal{V}$. Each element in $\mathcal{C}$ denotes a set of vertices with the size equal to or smaller than k. $m[v] \in \mathcal{C}$ denotes the set that the vertex v lies in. For example, collection $\mathcal{C}$ in iteration 3 of Fig. 7(e) has two elements, *i.e.,* {B, C, E} and {D, F}. We can access the vertex set {D, F} by either m[D] or m[F].

Algorithm 2 shows the overview of our algorithm GSS. $\mathcal{C}$ and $\forall v \in \mathcal{V}$ are initialized as $null$ at the beginning. We first estimate the benefit b of each edge using our *Estimation-based Benefit Model* (Line 3). Since we want to incorporate edges with large benefits into $G(V, E)$, we sort edges in $\mathcal{E}$ based on the benefit b in descending order (Line 4) and iterate on each of them (Line 5).

Given an edge (v, v'), we add its two endpoints into a vertex set in $\mathcal{C}$ (Line 6-14), so that the subgraph induced by the vertices in the set will have a large benefit. Now the problem is which vertex should be added to which vertex set. There are three cases. The first case is that both vertex v and v' do not appear in any vertex set in $\mathcal{C}$, we add $\{v, v'\}$ as a new vertex set to $\mathcal{C}$ and update m (Line 6-9). In the second case and third case, if $m[v] = null$ and $m[v'] \neq null$, we add v into $m[v']$. Otherwise, we add v' into $m[v]$ (Line 10-14). Please see Example 6 for details.

Then, we use $m[v_t]$ to denote the vertex set where v or v' is added. Next, we check whether the size of $m[v_t]$ is equal to k. If so, we can induce a new subgraph $G'(V', E')$ by vertices in $m[v_t]$ and compute its benefit $\mathcal{B}$ (Line 16-22).



(a) ERG with Weight $\mathcal{G}(\mathcal{V}, \mathcal{E}, w)$ (b) ERG with Benefit $\mathcal{G}(\mathcal{V}, \mathcal{E}, b)$ (c) G_1 (d) G_2

Iteration	1	2	3	4	5	6	7	...
Edge (v, v')	(B, E)	(B,C)	(D, F)	(G, H)	(A, E)	(G, I)	(G, J)	...
Collection $\mathcal{C}$	{B, E}	{B, C, E}	{B, C, E} {D, F}	{B, C, E} {D, F} {G, H}	{A, B, C, E} {D, F} {G, H}	{A, B, C, E} {D, F} {G, H, I}	{A, B, C, E} {D, F} {G, H, I, J}	...
Subgraph G								...

(e) Iterating Edges Phase

Fig. 7. Example of CQG Selection

After repeating $|\mathcal{E}|$ times, we can output the "most beneficial" subgraph $G(V, E)$ as CQG.

Example 6: [Example of CQG Selection.] Figure 7(a) is an ERG $\mathcal{G}(\mathcal{V}, \mathcal{E}, w)$. We first compute each edge's benefit based on our *Estimation-based Benefit Model* and get $\mathcal{G}(\mathcal{V}, \mathcal{E}, b)$ (Fig. 7(b)). Then we select the unvisited edge with maximum benefit in each iteration and put its two endpoints into a vertex set. Fig. 7(e) shows details in each iteration (suppose $k = 4$), in iteration 1, we select the edge (B, E). Since vertex B and E never appear in any vertex set of $\mathcal{C}$ (Case 1), we add {B, E} to $\mathcal{C}$. In iteration 2, we pick the edge (B, C). we find that vertex B exists in the set {B, E}, while vertex C does not appear any vertex set of $\mathcal{C}$ (Case 3). Thus, we set v_f = C, v_t = B and add vertex v_f into $m[v_t] = \{B, E\}$ (Line 14). We also update $m[v_f]$ by $m[v_f] = m[v_t]$ because v_f and v_t refer to the same vertex set (Line 15). We repeat the above steps until iteration 5, where we select the edge (A, E). We find that vertex A does not appear in any vertex set of $\mathcal{C}$, while vertex E exists in vertex set {B, C, E} (Case 2). Therefore, we set v_f = A, v_t = E, add vertex v_f into vertex set $m[v_t] = \{B, C, E\}$, and let $m[v_f] = m[v_t]$. Now the size of {A, B, C, E} equals to k. Hence, we construct a subgraph (*i.e.,* Fig. 7(c)) and compute its benefit. After iterating all edges, we construct two subgraphs (Fig. 7(c) and Fig. 7(d)) and we select the subgraph with maximum benefit as the output (Fig. 7(c)). □

Algorithm Analysis. In Algorithm 2, the time complexity of computing each edge's benefit is $\mathcal{O}(|\mathcal{E}|)$, and the time complexity of sorting $\mathcal{E}$ by benefit in descending order is $\mathcal{O}(|\mathcal{E}| \cdot log|\mathcal{E}|)$. Thus, the time complexity before iterating all edges is dominated by sorting phase *i.e.,* $\mathcal{O}(|\mathcal{E}| \cdot log|\mathcal{E}|)$. In iterating edge phase, the time complexity of maintaining the collection of vertex set $\mathcal{C}$ (Line 6-15) is constant. the time complexity of inducing subgraph $G'(V', E')$ and computing its benefit (Line 17-18) also is constant. Thus, the time complexity of iterating edge phase is $\mathcal{O}(\mathcal{E})$. Therefore, the overall time complexity of our algorithm is $\mathcal{O}(|\mathcal{E}| \cdot log|\mathcal{E}|)$.

Optimization for GSS. Based on our algorithm analysis, we could find that the bottleneck of our algorithm is the phase of sorting edges. Now, we introduce an optimization technique to prune the number of edges before the phase of sorting edges and an optimization technique to early terminate the phase of iterating edges. We use GSS$^+$ to denote the optimized version of GSS.

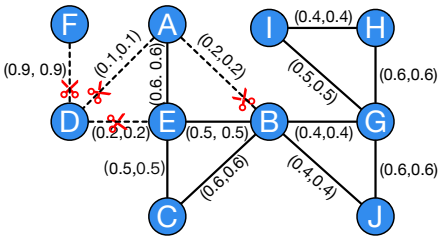


Fig. 8. Example of Edge Pruning

(1) Optimization by edges pruning. Since computing benefit of each edge and sorting edges by benefit is time-consuming in our algorithm GSS, an intuitive optimization method is to reduce the number of edges. Here, we propose a heuristic technique to prune edges. Intuitively, we would like that the subgraph contains more "uncertain" edges (*e.g.,* $weight \in [0.3, 0.7]$) because it usually provides much more "information" to enhance the quality of cleaning models (*e.g.,* EM model). Therefore, we could prune those edges with $weight \notin [0.3, 0.7]$ because those edges could be answered by the machine algorithm easily.

(2) Optimization by early termination. If we only want to compute the "best" subgraph, it is unnecessary for us to iterate all edges to find all subgraphs. Because we access the edge in descending order, the "best" subgraph is likely to be discovered in the first m subgraphs. Therefore, we can early terminate the loop if we convince that the "best" subgraph has already existed in our current finding m (*e.g.,* $m = 20$) subgraphs. We set $m = 20$ for our GSS$^+$ algorithm.

However, we should know that these pruning rules may let us miss the "best" subgraph. Nevertheless, we will show that GSS$^+$ with these pruning rules still performs well in our experiment in Section VII.

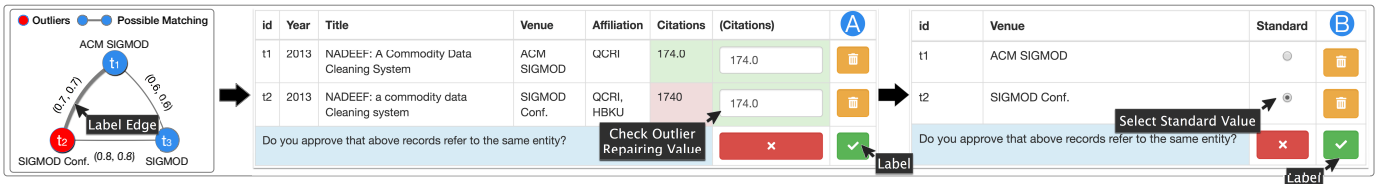
Example 7: [Example of Edge Pruning.] Figure 8 is an example of edge pruning. we can prune those edges with $weight \notin [0.3, 0.7]$. Thus, we can reduce the number of edges to save run time for computing benefit and sorting. □

Discussion. As mentioned earlier, compared with single questions, composite questions can provide more context. However, a user is often preferred to interact with a small (and thus succinct) graph, instead of a large graph. Hence, the number of vertices k is typically small (*e.g.,* $k <= 10$).

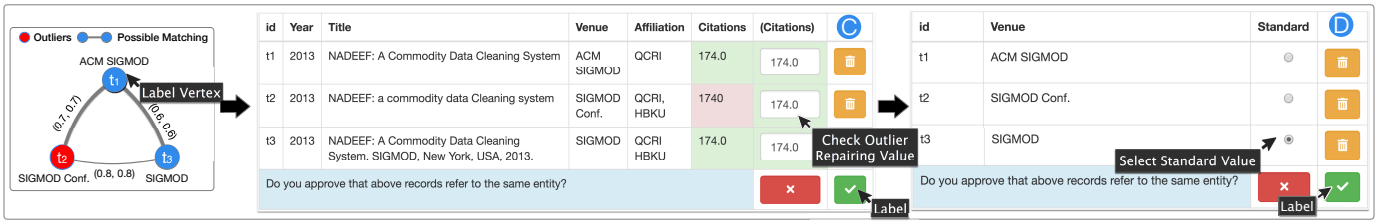
VI. User Operations on Composite Questions

In this section, we describe how the user can interact with the composite question to provide high-quality results in a user-friendly way. Roughly speaking, we consider two types of interaction – *label edge* and *label vertex*.

Label Edge. As introduced before, each edge links two possible matching entities, either tuple-level entities (*e.g.,* t_1 and t_2 in Table I) or attribute-level entities (*e.g.,* "ACM



(a) Example of Label Edge (t_1, t_2)



(b) Example of Label Vertex t_1

Fig. 9. Example of User Operations on a CQG

Dataset	(D1) DB Papers	(D2) NBA Players	(D3) Books
#-Attributes	6	17	17
#-Tuples	50,483	13,486	7,676
#-DistinctTuples	13,915	4,644	3,702
Missing Values%	15.1%	8.2%	9.2%
Outlier%	1.1%	1.3%	2.1%

TABLE IV
STATISTICS OF EXPERIMENT DATASETS

SIGMOD"↔"SIGMOD Conf."). Therefore, there are two possible operations on edge. The first one is to *confirm* the edge to label that its associated two vertices (*i.e.,* entities) are the same entity, while the second one is to *split* the edge to denote that the two entities are not the same entity. (*i*) If the user denotes that two tuple-level entities refer to the same entity, it will treat their attribute-level entities also as the same entity. As shown in Fig. 9(a), edge (t_1, t_2) links two tuple-level entities (t_1 and t_2) and attribute-level entities (the attribute for X). When the user clicks edge (t_1, t_2), our system will show details of (t_1, t_2) in table. If the system detects that there exists outliers (or missing values) in t_1 and t_2, it will suggest the outlier repairing values (or missing value imputation results). For example, in Fig. 9(a)–A, the user may first check the outlier repairing value and then check whether t_1 and t_2 refer to the same entity. Next, the user may click ✓ button to return the labeled answer. After that, the edge (t_1, t_2) will be confirmed. It depicts that entity t_1 and t_2, as well as t_1[Venue] and t_2[Venue], are matching. (*ii*) If the user clicks ✗ button, it will let the user further check the *attribute-level duplicates candidates* (*i.e., A*-questions) in Fig. 9(a)–B.

Label Vertex. Since each *label edge* operation can only provide a group of labeled answers, we also support *label vertex* operation. This type of operation can let the user label all edges linked to the same vertex. Therefore, this operation provides much more labeled answers in each interaction compared with the *label edge* operation. In Fig. 9(b), the vertex t_1 links two edges (t_1, t_2) and (t_1, t_3), namely two *tuple-level duplicates candidates* (*i.e., T*-questions) for EM model and two *A*-questions for standardizing attribute-level duplicates. For example, when the user clicks the vertex t_1 in Fig. 9(b), the system first constructs an EM cluster with three records – $\{t_1, t_2, t_3\}$ as shown in Fig. 9(b)–C. The user may exclude some records from this cluster by clicking 🗑 button and then label the remaining records as matching by clicking ✓ button. Our system will further show the *A*-questions in Fig. 9(b)–D for the user to label.

VII. EXPERIMENT

A. Experimental Setup

Datasets. We used three real-world datasets.

(D1) DB Papers. We crawled 50,483 papers (13,915 distinct papers), which were published on leading database conferences (*e.g.,* SIGMOD, VLDB, ICDE), from six sources (*e.g.,* DBLP and Google Scholar), with the schema (Title, Authors, Affiliation, Venue, Year, Citations).

(D2) NBA Players. We collected 13,486 records (4,644 distinct NBA Players) from three NBA communities (*e.g.,* https://www.basketball-reference.com/). Each record contains 17 attributes, *e.g.,* (Player, Position, Team, Nationality, University (Univ.), #Games, #Points, ...).

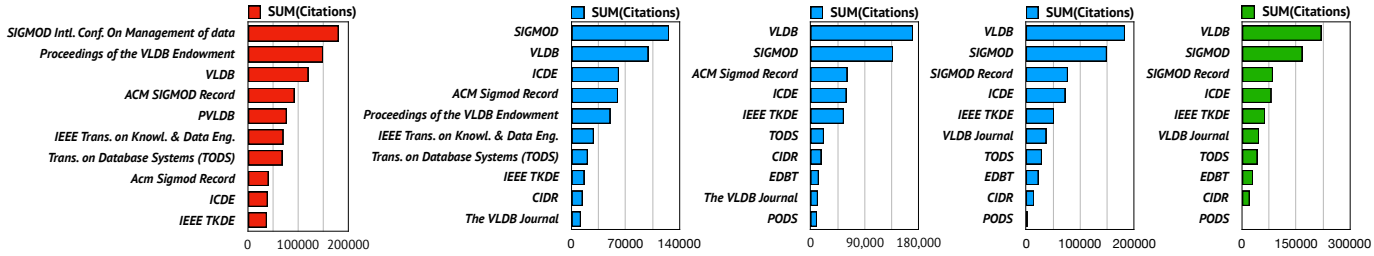
(D3) Books. A book ratings dataset was collected from two mainstream websites (*e.g.,* https://www.goodreads.com/). The dataset has 17 attributes, *e.g.,* (Name, Author, PubDate, Rating, NumofRating, Publisher (Publ.), Language (Lang), ...), and 7,676 records (3,702 distinct books).

In summary, the statistics of experimental datasets are shown in Table IV. We obtained the ground truth of tested datasets via crowdsourcing [23], [7].

Visualization Tasks. Table V shows 18 visualizations used in the experiment, which are selected to be representative, so as to cover most meaningful cases.

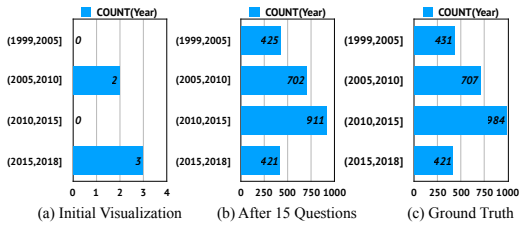
Algorithms. We compared the following algorithms: (*i*) GSS: our composite question graph (CQG) selection algorithm in the Section V-B. (*ii*) GSS$^+$: the optimized version of GSS. (*iii*) Random: it selects CQG randomly.

For evaluating the effectiveness of our CQG selection algorithms GSS and GSS$^+$, we have implemented the following algorithms as baselines: (*iv*) Branch and Bound (B&B) [21] is an algorithm to compute *Heaviest k-subgraph*, which can



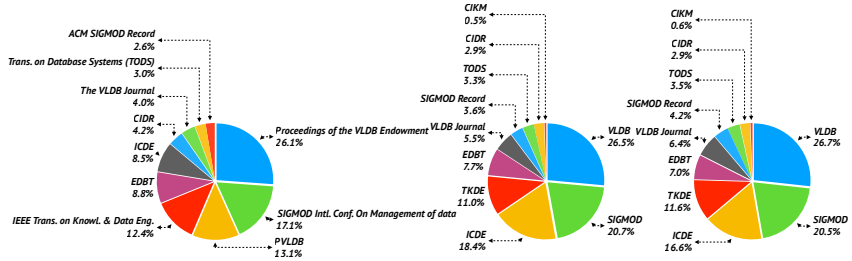
(a) Initial Visualization (b) After 5 Questions (c) After 10 Questions (d) After 15 Questions (e) Ground Truth

Fig. 10. Process of Visualization Improvement (Q1)



(a) Initial Visualization (b) After 15 Questions (c) Ground Truth

Fig. 11. Process of Visualization Improvement (Q7)



(a) Initial Visualization (b) After 15 Questions (c) Ground Truth

Fig. 12. Process of Visualization Improvement (Q8)

Q	V	X-axis	Y-axis	D	WHERE	TRANSFORM	S	L
1	B	Venue	S(Citations)	D1	–	GB (Venue)	D	10
2	B	Venue	A(Citations)	D1	–	GB (Venue)	–	–
3	P	Venue	C(Venue)	D1	–	GB (Venue)	D	10
4	B	Citations	C(Citations)	D1	–	BIN(Citations) BY INTERVAL 200	A	–
5	B	Year	C(Year)	D1	–	BIN(Year) BY INTERVAL 5	A	–
6	B	Venue	S(Citations)	D1	–	GB (Venue)	A	–
7	B	Year	C(Year)	D1	Year > 1999 and Venue = SIGMOD and Citations > 100	BIN(Year) BY INTERVAL 5	–	–
8	P	Venue	C(Venue)	D1	Year > 2009	GB (Venue)	D	10
9	B	Team	S(#Points)	D2	–	GB (Team)	D	10
10	B	Univ.	C(Univ.)	D2	Team = lakers	GB (Univ.)	–	–
11	B	Player	#Games	D2	–	–	D	10
12	B	#Points	C(#Points)	D2	Position = Forward	BIN(#Points) BY INTERVAL 5	A	–
13	P	Player	S(#Points)	D2	Position = Guard	GB(Player)	D	10
14	P	Publ.	C(Publ.)	D3	–	GB (Publ.)	D	10
15	B	Publ.	A(Rating)	D3	Lang = English	GB (Publ.)	D	10
16	B	Author	A(Rating)	D3	–	GB (Author)	D	10
17	B	Author	Rating	D3	Lang = English	–	D	5
18	B	Rating	C(Rating)	D3	–	BIN(Rating)) BY INTERVAL 1	–	–

TABLE V
VISUALIZATION TASKS. COLUMN Q FOR QUERY, V FOR VISUALIZATION (B: BAR, P: PIE), Y-AXIS (S: SUM, A: AVG: C: COUNT), TRANSFORM (GB: GROUP BY), S FOR SORT (D: DESC, A: ASC), L FOR LIMIT.

be used for CQG selection problem, and (*v*) α-B&B is an α-approximation version of B&B [21], where α is the approximation ratio.

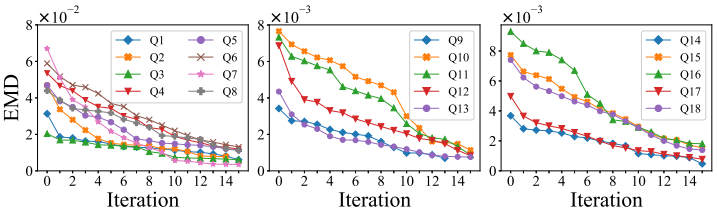
For comparing our CQG selection mechanism with a single question strategy, we have implemented an adaptive method that selects the "best" single question from repairing candidate sets as follows. (*vi*) Single: it selects a set of single questions to ask the user in each iteration. A single question is a pair of tuples to be matched (T-question $\in \mathbb{Q}_T$), a pair of attributes to be standardized (A-question $\in \mathbb{Q}_A$), a tuple with a missing value imputation result to be checked (M-question $\in \mathbb{Q}_M$), or a tuple with a possible outlier repairing result to be verified (O-question $\in \mathbb{Q}_O$). For a fair comparison, we treat a CQG as a unit cost and take a single question as $\frac{1}{m}$ unit cost (m equals to #-edges of a CQG). In each iteration, Single selects m questions from candidate sets. More specifically, $\frac{m}{4}$ single questions are selected from the candidate sets $\mathbb{Q}_T$, $\mathbb{Q}_A$, $\mathbb{Q}_M$, and $\mathbb{Q}_O$, respectively.

Experimental Environment. We implemented all algorithms by Python. We conducted all experiments on a Ubuntu server with 2.5 GHz 4 cores Intel CPU and 16GB RAM.

B. Experimental Results

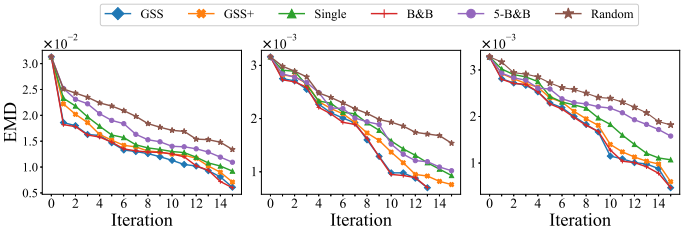
Exp-1: The End-to-End Evaluation. In the first set of experiments, we tested the end-to-end performance of VISCLEAN. We used the GSS to select the "best" CQG in each iteration to interact with the user. We set $budget$=15 for the number of iterations (one CQG in each iteration) and k=10 for the size of a CQG. We used the Earth Mover's Distance, EMD($Q(D)$,$Q(D_g)$), to quantify the visualization distance between current visualization ($Q(D)$) and the visualization ($Q(D_g)$) generated by the ground truth. The smaller the EMD is, the better the visualization quality is.

We first show a real running example to study our end-to-end performance. We use the visualization query Q1 in Table V, which visualizes the top-10 Venues ranking by total Citations. As shown in Fig. 10(a), it depicts the initial dirty visualization. The visualization distance between Fig. 10(a) and Fig. 10(e) (ground truth) is EMD(Fig.10(a),Fig.10(e))=0.031. After we asked 5 CQG questions, the visualization (Fig. 10(b)) is improved a lot. For example, many synonymous bars (*e.g.*, "VLDB" and "PVLDB") are merged, and some tuple-level duplicates are removed by the EM model. The visualization distance between the current visualization (Fig. 10(b)) and the ground truth is EMD(Fig.10(b),Fig.10(e)) = 0.014. After asking 10 CQG questions, the visualization (Fig. 10(c)) is significantly improved, which is already quite similar to the ground truth (Fig. 10(e)). For example, the top-3 Venues are the same as the



(a) DB Papers (b) NBA Players (c) Books

Fig. 13. EMD *vs.* Budget



(a) DB Papers (Q1) (b) NBA Players (Q9) (c) Books (Q14)

Fig. 14. EMD *vs.* Budget

ground truth. The Conference PODS first appears among the top-10 bars because VISCLEAN standardizes many attribute-level duplicates referring to the PODS entity, *e.g.,* standardize "PODS", "In Pods", and "PODS 2018". Therefore, the total Citations of PODS increased. After 15 iterations, VISCLEAN stops cleaning because our budget is consumed up, and we can see that the cleaned visualization (Fig. 10(d)) is very similar to the visualization generated from the ground truth (Fig. 10(e)). The EMD(Fig.10(d), Fig.10(e)) = 0.006. Note that, we need to ask hundreds of questions to clean the entire dataset to produce the ground truth.

From Fig. 10(d), we can learn that although the exact Citations of each Venue still have tiny differences with the visualization generated from ground truth, the order of each Venue is the same as the ground truth, which can support lots of visualization analysis scenarios.

We also depict the quality improvement of visualization Q7. This bar chart shows the total number of papers (citations$>$ 100) published at the SIGMOD conference every 5-year period after 1999. As shown in Fig. 11(a), the bar chart is very dirty compared with the ground truth (Fig. 11(c)) because of attribute-level duplicates (*e.g.,*SIGMOD and SIGMOD Conf.) in the selective predicate Venue = SIGMOD. This leads to that the number of papers within every five years cannot be counted accurately. However, VISCLEAN improves its quality a lot with 15 questions, as shown in Fig. 11(b). Similarly, as shown in Fig. 12, we can make a similar observation on the pie chart Q8, and we omit the discussion due to the space constraint.

Next, we report how the EMD($Q(D)$,$Q(D_g)$) decreases in the cleaning process. Fig. 13 shows that VISCLEAN can significantly reduce the EMD with a small budget. We take visualization Q1 in Fig. 13(a) as an example to illustrate how the EMD between current visualization and the ground truth change. In the initial status, the EMD is 0.031. In iteration 1, after the user answers a CGQ question, the EMD sharply decreases to 0.018 because lots of tuple-level duplicates and attribute-level duplicates are removed by data cleaning models. Then, the EMD steadily decreases in the interactive cleaning process. After 15 iterations, it produces the cleaned visualization (Fig. 10(d)), and its EMD reduces to 0.006. We can see that the visualization quality of Q1 is significantly improved from the initial status (Fig. 10(a)). Similarly, the EMD gradually decreases in the interactive cleaning process on visualization Q2. After 14 iterations, it terminates the cleaning process because the user satisfies the cleaning results. Similar observations can be derived by experiments on other datasets (Fig. 13(b) and Fig. 13(c)). We omit to discuss due to space limitations.

We make two observations from this group of experiments: (*i*) VISCLEAN can significantly improve the visualization quality with a small budget (*i.e.,* #-iterations).
(*ii*) Cleaning different visualizations generated from the same dataset needs different budgets.

Exp-2: Effectiveness of CGQ Selection. The main purpose in this group of experiments is to test: (*i*) whether the composite questions are more effective than asking a group of single questions in isolation, and (*ii*) whether our CQG selection algorithms are effective enough compared with the exact CQG selection algorithm B&B and its α-approximation version α-B&B ($\alpha = 5$).

We set k = 10 for the size of a CQG in each iteration and set $budget$ = 15 for the #-iterations.

Figure 14 shows that all algorithms we tested can reduce the EMD in 15 iterations. We make the following observations: (*i*) The algorithms asking composite questions, GSS, GSS$^+$, and B&B, are better than the algorithm Single asking a group of single questions in isolation. The reason is that composite questions can provide more information to improve the quality of cleaning models, and thus enhance the visualization quality. (*ii*) The effectiveness of our CQG selection algorithms, GSS and GSS$^+$, are similar (but slightly worse) to the exact CQG selection algorithm B&B. However, GSS and GSS$^+$ algorithms outperform 5-B&B by 30%-60%. Experimental results verify the effectiveness of GSS and GSS$^+$.

Next, we evaluated the effectiveness of our composite questions mechanism and single questions strategy in terms of the human cost. First, we recruited 20 participants from the CS Department as real users to participate in this group of experiments. All participants know visualization and have the necessary skills of interacting with the graph and table, but none of them is familiar with data cleaning. Before the experiment, we asked each participant to browse experimental datasets and made a demonstration on a composite question and a group of single questions for each dataset in our system. We then asked each participant to answer 15 iterations of composite questions and 15 iterations of single questions on three datasets. We recorded the interaction time of each question in each iteration. Finally, we calculated the average user time across all participants in each iteration.

Figure 15 shows the average user time of answering a CQG question and a group of single questions in each iteration. Overall, the user spends less time answering a CQG question

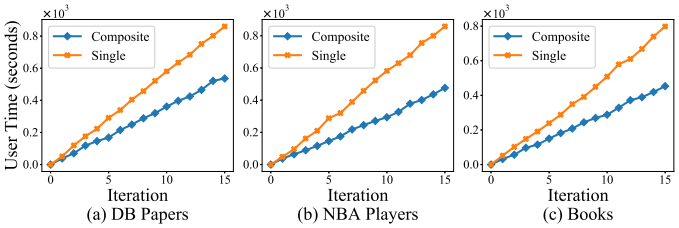


Fig. 15. Average User Time *vs.* Budget

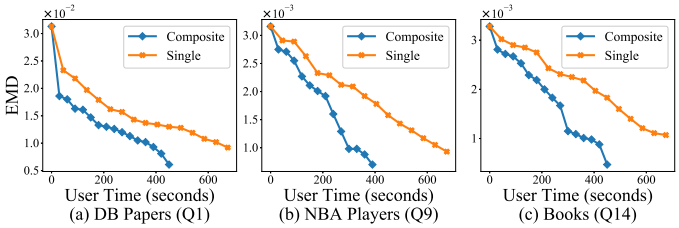


Fig. 16. Average User Time *vs.* EMD (Budget = 15)

Tasks	WrongLabel% (W%)			Completeness% (C%)		
	0%	5%	10%	**100%**	95%	90%
Q1	**15**	17	19	**15**	16	18.5
Q2	**14**	16	18.5	**14**	15.5	17
Q3	**15**	17.5	19.5	**15**	17	19

TABLE VI
#-QUESTIONS ASKED UNDER DIFFERENT SETTINGS (AVERAGE)

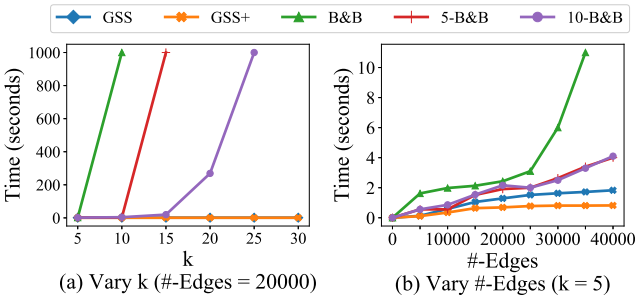


Fig. 17. The Efficiency of CQG Selection

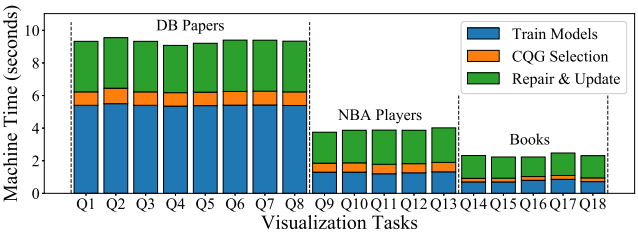


Fig. 18. Average Machine Time in Each Iteration

compared with a group of single questions in each iteration. For example, in Fig. 15(a), the user spends about 520 seconds answering 15 iterations of CQG questions, while spends about 860 seconds answering 15 iterations of a group of single questions. Similarly, Fig. 16 reports how the EMD of three queries decreases according to the user time. It shows that the EMD of the composite questions mechanism decreases faster than the EMD of the single questions strategy. For example, in Fig. 16(b), when the user time is 400 seconds, the EMD of the composite questions mechanism drops to 0.0007 while the EMD of the single questions strategy only declines to 0.0018.

Figures 15 and 16 tell us that the composite questions mechanism saves about 40% user time compared with the single questions strategy. The reason is that the user can answer composite questions more easily and precisely compared with a set of single questions in isolation.

Exp-3: Impact of User Input. In this part, we evaluated the performance of VISCLEAN with incorrect and incomplete user answers, respectively. To this end, we produce some errors among the former answers of experts in Exp-1. More specifically, for testing incorrect user inputs, we *randomly* injected 5% and 10% wrong labels into *experts*' answers to a CQG. For evaluating incomplete user input, we *randomly* sampled 95% and 90% labels from their answers. Finally, we repeated this set of experiments three times to compute the average results. The results are shown in Table VI. For example, in task Q1, when we injected 5% wrong labels, VISCLEAN only asks 17 CQG to achieve a very similar high-quality result as the one without wrong labels, with only two more CQG questions. What's more, when Completeness%=95%, VISCLEAN costs only one more CQG than that of Completeness%=100%. Therefore, our approach can tolerate reasonably incorrect and incomplete user inputs.

Exp-4: Efficiency of CQG Selection. We also compared the efficiency of our CQG selection algorithms GSS and GSS$^+$ with two baselines B&B and $\alpha-$B&B.

We first fixed the size of the ERG (#-edges = $20,000$) and varied the size of CQG (k from 5 to 30).

Figure 17(a) reports the results and tells us the followings: (*i*) Our algorithm GSS and GSS$^+$ significantly outperform the B&B, 5-B&B, and 10-B&B; (*ii*) It is hard for B&B, 5-B&B, and 10-B&B to efficiently compute a CQG when $k > 10$; and (*iii*) GSS$^+$ outperforms GSS by $30\% - 40\%$.

Next, we set $k = 5$ and varied #-edges of the ERG from $5,000$ to $40,000$. As shown in Fig. 17(b), GSS and GSS$^+$ outperform the B&B, 5-B&B, and 10-B&B. Besides, thanks to optimization techniques (*i.e.,* edges pruning and early termination), GSS$^+$ is faster than GSS by $30\% - 40\%$.

We computed the average machine time for each component of VISCLEAN. Fig. 18 shows the average machine time in each iteration of each visualization task. It shows that *Train Models* takes the bulk of the time because it needs to train (or fine tune) the EM model for removing tuple-level duplicates and maintain kNN for repairing outliers and missing values.

VIII. RELATED WORK

Progressive Visualization aims to compute an approximate visualization at interactive speeds and then continues to improve its quality [26], [18]. A recent study [36] suggests that (i) the users can discover more insights through progressive visualization and (ii) progressive visualization is a promising approach to achieve scalability in the visual analysis system. Our work is the first to do progressive visualization from a dirty data perspective, not from a performance perspective.

Generic Data Cleaning have been widely studied from rule-based detection algorithms [1], [15], [34], quantitative error detection algorithms that expose outliers in the data [27], to record linkage and de-duplication algorithms for detecting duplicate data records (see [11] for surveys) such as Tamr. There are many tuple-level de-duplication techniques [31], [19], [8], [13], [22], and attribute-level data transformation for various data types [10], [35], [30], [37]. We used [19] to detect and remove tuple-level duplicates and used [10] to detect and remove attribute-level duplicates candidates.

Task-driven Data Cleaning. ActiveClean [20], an iterative cleaning framework, aims to clean a small subset of data to achieve the high quality of a model similar to the case if the data was fully cleaned. Altwaijry et al. study the problem of query-driven data cleaning [4], where given a query, they clean data relevant to the query. However, they focus on the entity resolution problem and do not consider to clean other types of errors (*e.g.,* outliers). Besides, they do not consider whether the data errors affect the quality of visualizations.

Data Cleaning for Visualization. Tableau Prep (https://tableau.com/products/prep) provides operations such as filter, split, rename to clean and transform dataset. OpenRefine (http://openrefine.org/) supports de-duplication and simple string transformation. Data Wrangler [17] provides some simple string transformation such as replacing, string splitting, etc. DataXFormer [3] can leverage the user-provided examples to generate string transformations from web tables and knowledge bases. However, these tools need to heavily involve the user to clean the dataset.

IX. Conclusion

In this paper, we have studied the problem that is progressively turning bad visualizations into good ones by interactive data cleaning. We have presented an errors and repairs graph to model all possible errors and repairs. We have also proposed to use the composite question, *i.e.,* a group of single questions to be treated as one question, to ask the user. We have further presented a novel GUI for easy user interaction. Finally, we have devised an effective and efficient composite question selection algorithm that asks the most beneficial question to generate the best visualization results. Experimental results on real datasets have shown that our methods can generate high-quality visualizations by asking a small number of questions. Note that, a potential problem is whether these repairs can be directly pushed back to clean the database gradually. In most practical settings, the users treat these tables as materialized views for their downstream applications. Instead of directly applying these updates, they should be typically used as suggestions for the DBA to approve.

Acknowledgement. This work was supported by 973 Program of China (2015CB358700), NSF of China (61632016, 61472198, 61521002, 61661166012), Huawei, and TAL.

References

[1] Z. Abedjan, C. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker. Temporal rules discovery for web data cleaning. *PVLDB*, 9(4), 2015.

[2] Z. Abedjan, X. Chu, and D. D. et.al. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.

[3] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, and et. al. Dataxformer: A robust transformation discovery system. In *ICDE*, 2016.

[4] H. Altwaijry, D. V. Kalashnikov, and S. Mehrotra. Query-driven approach to entity resolution. *PVLDB*, 6(14):1846–1857, 2013.

[5] M. Bergman, T. Milo, S. Novgorodov, and W. C. Tan. Query-oriented data cleaning with oracles. In *SIGMOD*, 2015.

[6] C. Binnig, L. D. Stefani, T. Kraska, E. Upfal, E. Zgraggen, and Z. Zhao. Toward sustainable insights, or why polygamy is bad for you. In *CIDR*.

[7] C. Chai, J. Fan, G. Li, J. Wang, and Y. Zheng. Crowdsourcing database systems: Overview and challenges. In *ICDE 2019*.

[8] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD, 2016*.

[9] X. Chu, J. Morcos, , and et.al. KATARA: a data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, 2015.

[10] D. Deng, G. Li, W. Tao, and et.al. Unsupervised String Transformation Learning for Entity Consolidation. In *ICDE*, 2019.

[11] F. N. et al. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[12] J. H. et al. In pursuit of error: A survey of uncertainty visualization evaluation. *IEEE Trans. Vis. Comput. Graph.*, 25(1), 2019.

[13] C. C. et.al. A partial-order-based framework for cost-effective crowdsourced entity resolution. *VLDB J.*, 2018.

[14] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *SIGMOD*, 2011.

[15] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *VLDBJ*, 21(2):213–238, 2012.

[16] Y. Jiang, G. Li, J. Feng, and W. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.

[17] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *CHI*, 2011.

[18] A. Kim, E. Blais, A. G. Parameswaran, and et.al. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 2015.

[19] P. Konda, S. Das, P. S. G. C., and A. D. et.al. Magellan: Toward building entity matching management systems. *PVLDB*, 2016.

[20] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and et al. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB*, 2016.

[21] M. Letsios, O. D. Balalau, M. Danisch, E. Orsini, and M. Sozio. Finding heaviest k-subgraphs and events in social media. In *ICDM Workshops,*.

[22] G. Li and a. Chengliang Chai et. CDB: optimizing queries with crowd-based selections and joins. In *SIGMOD, 2017*.

[23] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE Trans. Knowl. Data Eng.*, 2016.

[24] Y. Luo, X. Qin, N. Tang, and G. Li. DeepEye: Towards Automatic Data Visualization. In *ICDE*, 2018.

[25] Y. Luo, X. Qin, N. Tang, G. Li, and X. Wang. Deepeye: Creating good data visualizations by keyword search. In *SIGMOD*, 2018.

[26] M. Procopio, C. Scheidegger, E. Wu, and R. Chang. Selective wander join: Fast progressive visualizations for data joins. *Informatics*, 2019.

[27] N. Prokoshyna, J. Szlichta, F. Chiang, R. J. Miller, and D. Srivastava. Combining quantitative and logical data cleaning. *PVLDB*, 9(4), 2015.

[28] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*, 2000.

[29] A. Satyanarayan, D. Moritz, and et al. Vega-lite: A grammar of interactive graphics. *IEEE Trans. Vis. Comput. Graph.*, 23(1), 2017.

[30] R. Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *PVLDB*, 2016.

[31] R. Singh, V. V. Meduri, A. K. Elmagarmid, S. Madden, P. Papotti, J. Quiané-Ruiz, A. Solar-Lezama, and N. Tang. Synthesizing entity matching rules by examples. *PVLDB*, 11(2):189–202, 2017.

[32] N. Tang, E. Wu, and G. Li. Towards democratizing relational data visualization. In *SIGMOD*, pages 2025–2030, 2019.

[33] M. Vartak, S. Rahman, and et.al. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 2015.

[34] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468, 2014.

[35] Y. Wang and Y. He. Synthesizing mapping relationships using table corpus. In *SIGMOD*, pages 1117–1132, 2017.

[36] E. Zgraggen, A. Galakatos, and et.al. How progressive visualizations affect exploratory analysis. *IEEE Trans. Vis. Comput. Graph.*, 2017.

[37] E. Zhu, Y. He, and S. Chaudhuri. Auto-join: Joining tables by leveraging transformations. *PVLDB*, 10(10):1034–1045, 2017.